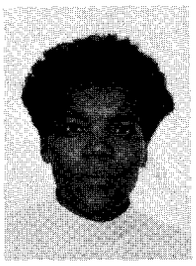
Modélisation compacte de cartes à microprocesseur : Calcul de la longueur de test aléatoire

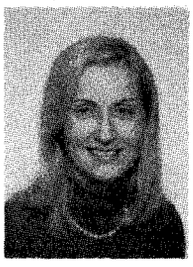
Compact modeling tool for microprocessor boards: random test length calculation



Zineb ABAZI

USTHB, Institut d'Électronique, BP 9, DAR EL BEIDA, ALGÉRIE.

Zineb Abazi, Ingénieur en Électronique, diplômée de l'École Nationale Polytechnique d'Alger (1983), a obtenu le titre de Docteur d'Université à l'Institut National Polytechnique de Grenoble en mai 1987. Ses travaux de recherche, effectués au Laboratoire d'Automatique de Grenoble de 1984 à 1987, ont porté sur le test aléatoire des cartes logiques. Depuis octobre 1987, elle est en Algérie où elle enseigne l'Automatique à l'Université Scientifique et Technologique d'Alger.



Pascale THEVENOD-FOSSE

Laboratoire d'Automatique et d'Analyse des Systèmes, 7, avenue du Colonel-Roche, 31077 TOULOUSE CEDEX, FRANCE.

Pascale Thévenod-Fosse, Ingénieur en Informatique et Mathématiques Appliquées diplômée de l'Institut National Polytechnique de Grenoble (1975) et Docteur ès-Sciences Physiques (1983), est Chargée de Recherche au CNRS. Elle a travaillé au Laboratoire d'Automatique de Grenoble de 1975 à 1986 où ses travaux ont porté essentiellement sur le test aléatoire de matériel (composants intégrés, cartes logiques). Depuis janvier 1987, elle a rejoint à Toulouse le Laboratoire d'Automatique et d'Analyse des Systèmes du CNRS. Ses recherches actuelles concernent le test et la fiabilité du logiciel.

RÉSUMÉ

Le test aléatoire est une méthode de test fonctionnel hors ligne. Il consiste à appliquer une séquence de vecteurs aléatoires aux entrées de la carte sous test. Le but des recherches théoriques est de calculer le nombre de vecteurs nécessaires, appelé *longueur de test*, pour garantir une qualité de détection donnée. Dans des travaux antérieurs, des modèles markoviens ont été proposés pour analyser le comportement d'une carte à microprocesseur contenant une mémoire morte. Cet article propose un nouveau modèle appelé *modèle compact*, qui se déduit des précédents mais contient un nombre plus restreint d'états. Une carte défectueuse est ensuite représentée par une *chaîne de Markov absorbante* dérivée du modèle compact, et la longueur de test se calcule par simple résolution du système markovien associé. La méthode est illustrée sur un exemple de carte hypothétique. Puis on donne des résultats numériques obtenus pour une carte réelle.

MOTS CLÉS

Test aléatoire, cartes logiques, chaînes de Markov, longueur théorique de test.

SUMMARY

Functional random testing consists in applying a sequence of random input patterns to a board under test. The research aim is to calculate the length of the test sequence *for a given detection quality. Previous work has used Markov models to analyse the behavior of a microprocessor board containing a Read Only Memory. Today's paper presents a new model, called* compact model, *deduced from the previous ones but with a smallest state number. Then, one describes a faulty card as an* absorbing Markov chain *constructed from the compact model, and the test length is calculated by solving the associated Markov system. The method is illustrated with a hypothetical example. One also gives numerical results obtained for a real board.*

KEY WORDS

Random testing, logical boards, Markov chains, theoretical test length.

1. Introduction

Pour les cartes logiques, les équipements actuels de test automatique peuvent être regroupés en deux grandes familles : les testeurs fonctionnels et les testeurs *in situ* [1-3]. Le *test in situ* (en anglais, *in circuit*) procède composant par composant en isolant électriquement chaque circuit pour le tester à part, tandis que le *test fonctionnel* teste la carte entièrement assemblée. En pratique, ces deux techniques sont complémentaires [4, 5]. Dans les deux cas, les vecteurs de test appliqués aux entrées de la carte sous test peuvent être soit définis de façon déterministe, soit engendrés par un générateur aléatoire (ou pseudo-aléatoire). Bien que les testeurs déterministes soient d'usage courant de nos jours, le test aléatoire est une solution intéressante pour les cartes logiques [6].

Une stratégie complète de *test aléatoire* doit comporter deux étapes. Un test fonctionnel du type « bon/pas bon », rapide et peu coûteux, permet d'abord de savoir si la carte est ou non défectueuse (étape 1). Lorsque le fonctionnement s'est révélé incorrect, un test *in situ* est ensuite plus performant pour localiser le ou les composant(s) responsable(s) du comportement défectueux observé globalement. Cette seconde étape peut se faire de façon aléatoire en vérifiant individuellement chacun des boîtiers montés sur la carte. Les résultats relatifs au test aléatoire de circuits, publiés antérieurement [7-13], sont alors directement utilisables. Cet article concerne donc seulement le test *fonctionnel* (étape 1), avec pour objectif une détection rapide des cartes défectueuses. On se limite aux cartes contenant un microprocesseur (μP) et une mémoire morte (ROM) dans laquelle est écrit un programme d'application (voir *fig.* 1).

Une *expérience de test aléatoire fonctionnel* consiste à appliquer une séquence d'instructions aléatoires ou pseudo-aléatoires avec des données (pseudo-)aléatoires aux entrées de la carte sous test. Les signaux observés en sortie sont comparés avec ceux d'une carte supposée bonne [6, 11], et une erreur est détectée lorsqu'ils diffèrent. Les instructions présentées aux entrées de la carte sont choisies aléatoirement parmi l'ensemble des instructions du μP monté sur la carte. La séquence des vecteurs générés forme alors un programme, appelé *programme de test*, qui est *syntaxiquement correct* mais qui n'a pas de signification sémantique précise [8-10]. Le problème théorique auquel on s'intéresse est la définition d'une méthode permettant

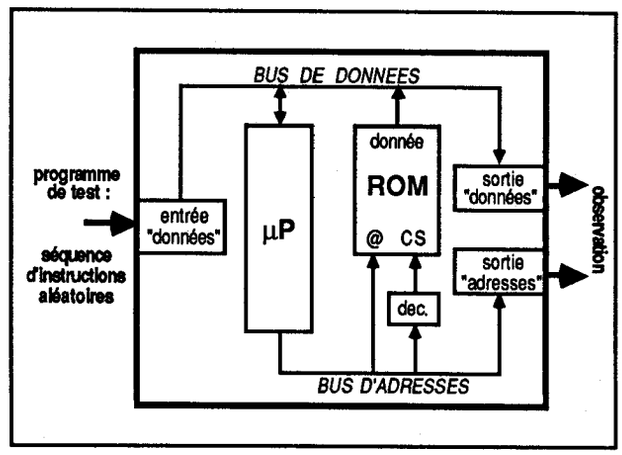


Fig. 1. – Principe du test aléatoire fonctionnel pour une carte contenant un microprocesseur (μP) et une mémoire ROM.

d'évaluer le nombre d'instructions aléatoires nécessaires pour révéler toute faute appartenant à un ensemble donné (*hypothèses de fautes*) avec une probabilité (*qualité de détection*) donnée. Le travail présenté est basé sur les chaînes de Markov [14]. On propose un outil de modélisation permettant de décrire et d'analyser le comportement d'une carte pendant une expérience de test aléatoire.

Le paragraphe 2 concerne les *hypothèses de fautes* prises en compte dans l'étude. On rappelle une propriété générale du test aléatoire, et son application aux cartes logiques qui justifie la démarche suivie dans l'analyse théorique. Le paragraphe 3 est ensuite consacré à la *modélisation d'une carte non défectueuse.* L'approche proposée conduit à une représentation par une chaîne de Markov, appelée modèle compact (C-modèle), qui contient un nombre raisonnable d'états. Partant d'un modèle initial détaillé de la carte, le C-modèle s'obtient par regroupements successifs d'états, ce qui permet de réduire progressivement la complexité (nombre d'états) de la chaîne. Le modèle qui décrit le comportement d'une *carte affectée d'une faute f*, est présenté au paragraphe 4. C'est une *chaîne de Markov absorbante* déduite du C-modèle de la carte non défectueuse. L'état absorbant correspond à la détection. Il est atteint dès qu'un signal erroné est observé en sortie de la carte. La *longueur du programme de test* (nombre d'instructions aléatoires) se calcule en résolvant le système markovien associé à la chaîne absorbante. La méthode est illustrée par son application à une carte hypothétique avec une

ROM de 10 mots de 5 bits contenant le programme d'application P donné dans la figure 2. P, composé de 6 instructions, correspond à une boucle de retard. En conclusion, les résultats numériques obtenus pour une *carte réelle* sont présentés au paragraphe 5. Ils montrent que la méthode conduit à un temps de test raisonnable d'une minute.

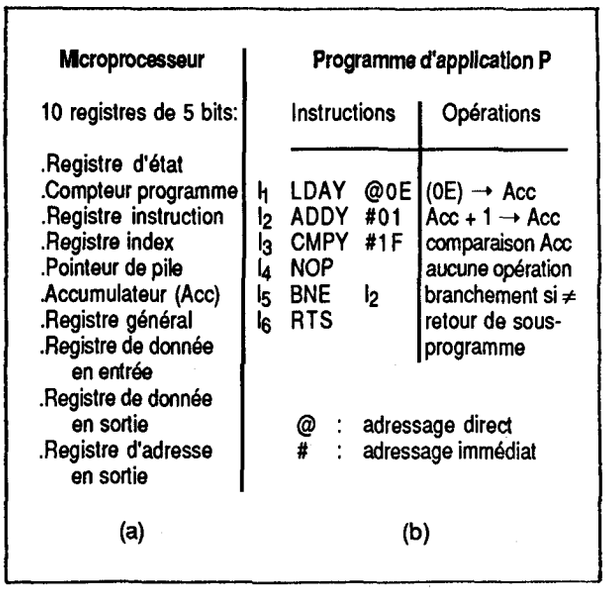


Fig. 2. – **Carte hypothétique.** (*a*) Registres du μP hypothétique commandé par 26 instructions (6 codes opérations sont invalides). (*b*) Programme d'application écrit en ROM : 6 instructions $I_1, \ldots, I_6$.

2. Résultats préliminaires

2.1. Terminologie

Faute, erreur et défaillance sont trois termes utilisés pour décrire un système défectueux [15]. Dans le cas d'une carte, le collage d'un bit en ROM dû à un court-circuit entre deux connexions est un exemple de *faute f*. La conséquence de cette faute *f* est la présence permanente d'une valeur erronée v_f dans un mot de la mémoire. On dit qu'il y a une *erreur* latente v_f en ROM. Cette erreur est activée lorsque le μP lit le mot contenant v_f. L'activation de v_f peut mettre une ou plusieurs valeurs fausses $V_f (V_f = v_f$ ou $V_f \neq v_f)$ dans un ou plusieurs registres internes du μP. L'activation des erreurs V_f peut ensuite créer d'autres erreurs, etc. Quand l'une d'entre elles est propagée en sortie de la carte, il y a *défaillance*. Pendant une expérience de test aléatoire, la première défaillance due à la présence d'une faute *f* correspond à la détection de la faute.

2.2. Propriété du test aléatoire

Une propriété générale du test aléatoire, indépendante du circuit ou système à tester et des hypothèses de fautes, a été démontrée et utilisée dans différents travaux antérieurs relatifs au test de circuits [7-11, 13].

Dans le cas des cartes où la longueur de la séquence de test se mesure par le nombre d'instructions aléatoires dans le programme de test, cette propriété s'exprime de la façon suivante.

Propriété 1 : Soit L_f la longueur du programme de test nécessaire pour détecter une faute *f* avec une incertitude de détection Q_D donnée (Q_D est la probabilité de ne pas détecter *f*). La longueur L du programme de test nécessaire pour détecter, avec une incertitude de détection Q_D fixée, toute faute *f* appartenant à un ensemble de fautes pré-défini F est :

$$L = \max L_f, \quad f \in F. \quad \square$$

Les fautes *f* qui nécessitent la longueur maximale $(L = L_f)$ sont appelées *fautes les plus difficiles à détecter*, en terme de probabilité, dans l'ensemble F.

La propriété 1 permet de diminuer le nombre de fautes à étudier dans un ensemble initial F qui correspond aux hypothèses de fautes. En effet, si les fautes les plus difficiles à détecter appartiennent à un sous-ensemble connu $F_D (F_D \subset F)$, il suffit de calculer L_f pour les fautes $f \in F_D$, et on en déduit L.

2.3. Hypothèses de fautes

Dans cette étude, on s'intéresse aux fautes qui affectent la ROM et le μP montés sur la carte. En effet, pour ces deux types de composants, des modèles de fautes sont définis dans la littérature et utilisés de façon « classique », ce qui n'est pas le cas pour les autres éléments (amplificateurs de bus, décodeurs, ...). Notons que cela ne signifie pas que ces autres éléments ne sont pas testés. Ils le sont, mais avec une incertitude de détection que l'on ne chiffre pas de façon théorique. L'ensemble F des fautes incluses dans les *hypothèses* se compose donc de deux sous-ensembles : $F = R \cup m$ où R désigne les fautes dans la ROM, et *m* celles dans le μP.

Pour les *mémoires mortes*, des études antérieures [7] ont montré que les fautes les plus difficiles à détecter sont les collages simples à 0 (un seul bit ne peut pas prendre la valeur 1) ou à 1 (un seul bit ne peut pas prendre la valeur 0). D'après la propriété 1, les fautes à étudier dans R se restreignent donc au sous-ensemble R′ des collages simples. Par exemple, les collages multiples (plusieurs bits ont une seule valeur possible) n'appartiennent pas à R′. Notons que chaque faute $f \in R'$ ne peut introduire qu'un seul mot erroné en ROM.

Un *microprocesseur* se décompose en deux parties : la partie opérative et la partie commande. On considère ici le modèle fonctionnel de fautes étudié dans des travaux détaillés sur le test aléatoire de μP [8, 9]. L'ensemble *m* est formé de quatre sous-ensembles, $m = \{r \cup o \cup d \cup i\}$, caractérisés comme suit. Les fautes dans la partie opérative affectent soit les registres (sous-ensemble de fautes noté *r*), soit les opérateurs (sous-ensemble *o*). Les fautes dans la partie commande affectent soit la fonction décodage d'adresse (sous-ensemble *d*), soit la fonction décodage d'instructions (sous-ensemble *i*). Les sous-ensembles *r* et *o* sont détaillés dans [8], et les sous-ensembles *d* et *i* dans [9] où l'on montre que la faute la plus difficile

à détecter dans m est certainement une faute f affectant un opérateur. Dans l'ensemble m, il suffit donc d'étudier les fautes appartenant au sous-ensemble o.

En conclusion, les résultats antérieurs relatifs aux ROM et aux µP montrent que la faute la plus difficile à détecter dans l'ensemble $F = \{R \cup m\}$ appartient au sous-ensemble $F' = \{R' \cup o\} \subset F$. Les travaux théoriques sur le test des cartes ont donc pour objectif d'analyser les fautes appartenant au sous-ensemble F'.

2.4. Étude comparative des fautes

L'ensemble F' contient trop de fautes pour qu'il soit raisonnable d'envisager le calcul de toutes les longueurs de test L_f, $\forall f \in F'$. Par une étude comparative des fautes appartenant à F', on a donc cherché à définir un sous-ensemble restreint de fautes pour lesquelles les calculs sont nécessaires. Ces comparaisons, détaillées dans [16, 17], sont liées aux notions de *commandabilité* (activation d'une erreur latente) et d'*observabilité* (propagation d'une valeur fausse en sortie de la carte) des fautes. Les résultats obtenus conduisent à la proposition suivante.

Proposition 1 : La *faute* $f \in F'$ *la plus difficile à détecter* est un collage simple dans un mot de la ROM (f affecte une seule instruction I_i du programme d'application) qui remplit les trois conditions suivantes :

C1. la modification de I_i par f ne change pas le séquencement du programme d'application;

C2. I_i se trouve en début du programme d'application et n'appartient pas à une boucle;

C3. aucune instruction exécutée après I_i dans le programme d'application ne permet la détection de f. □

Soit F_D l'ensemble des fautes remplissant ces trois conditions. La proposition définit un *sous-ensemble* $F_D \subset F'$ *qu'il suffit d'étudier*. Elle s'est trouvée vérifiée dans le cas des différents programmes d'application pour lesquels les longueurs de test L_f associées à des fautes $f \notin F_D$ (fautes en ROM et dans le µP) ont été calculées [17]. Tous ces résultats numériques confirment que la faute la plus difficile à détecter appartient à F_D ($F_D \subset R' \subset F' \subset F$). En pratique, il s'avère donc raisonnable de considérer comme vraie la proposition 1, pour se limiter à l'étude des fautes du sous-ensemble F_D à l'aide de la méthode présentée dans les paragraphes qui suivent.

Exemple de la carte hypothétique

Soit h le collage à 1 du bit de poids faible dans le second mot de la ROM contenant le programme P (*fig. 2 b*). h affecte l'instruction I_1 en changeant l'adresse 0E en $v_h = 0F$. Lorsque l'erreur v_h est activée, le µP lit un opérande à l'adresse fausse 0F et met donc une valeur fausse dans l'accumulateur (Acc). Les autres instructions I_j, $2 \leqq j \leqq 6$, sont inchangées. Donc la condition C1 est vraie. Comme aucune instruction ne propage le contenu de Acc, C3 est vérifiée. P commence par I_1 qui n'est pas dans une boucle $\Rightarrow$ C2 vraie. Par conséquent, le collage h est un exemple de faute qui appartient à F_D.

3. Modélisation d'une carte non défectueuse

3.1. Principe

Au début d'une expérience de test, le µP monté sur la carte exécute des instructions appliquées aux entrées de la carte, engendrées par un générateur aléatoire externe à la carte. Cette séquence d'instructions forme le *programme de test*. Mais la présence de la ROM sur la carte entraîne une exécution possible d'une ou plusieurs des instructions qu'elle contient, qui constituent le *programme d'application*. En effet, lorsqu'une instruction aléatoire correspond à un branchement en ROM, le µP va lire en mémoire la prochaine instruction à exécuter. Il continue l'exécution en séquence des instructions écrites en ROM jusqu'à ce que l'une d'entre elles provoque un branchement à une adresse externe à la carte. Les instructions suivantes viennent alors à nouveau du générateur jusqu'au prochain saut aléatoire en ROM, etc. Cette alternance entre les deux programmes peut se représenter par une chaîne de Markov à deux états comme le montre la figure 3. La carte est dans l'état TP lorsque le µP exécute une instruction du *P*rogramme de *T*est, et dans l'état AP quand l'instruction appartient au *P*rogramme d'*A*pplication.

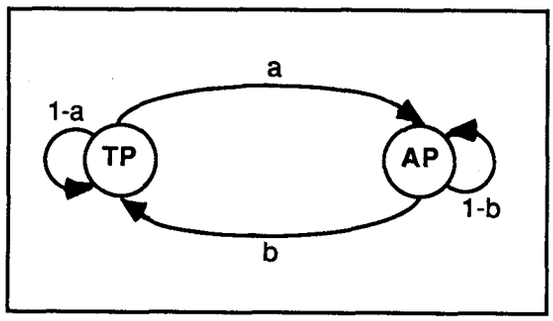


Fig. 3. – Principe de la modélisation d'une carte non défectueuse.

La probabilité, notée a sur la figure 3, d'exécuter une des instructions en ROM après une instruction externe dépend des probabilités des instructions de branchement dans le programme de test. La probabilité, notée b sur la figure 3, de revenir exécuter le programme de test après une instruction en ROM dépend du programme d'application qui est figé. Le calcul de b nécessite une modélisation plus détaillée du contenu de la ROM.

Des *modèles exacts*, définis dans une première étude [6], sont rappelés au paragraphe 3.2. Ils ne nécessitent aucune approximation dans le calcul des probabilités des transitions, mais ils conduisent à des chaînes de Markov avec un nombre d'états très élevé dans le cas de cartes réelles. C'est pourquoi, on propose ensuite de nouveaux modèles appelés *modèles compacts* (§ 3.3). Ils se déduisent des modèles précédents grâce à deux règles de réduction qui permettent de diminuer progressivement le nombre d'états, mais qui correspondent dans certains cas à des calculs approximatifs.

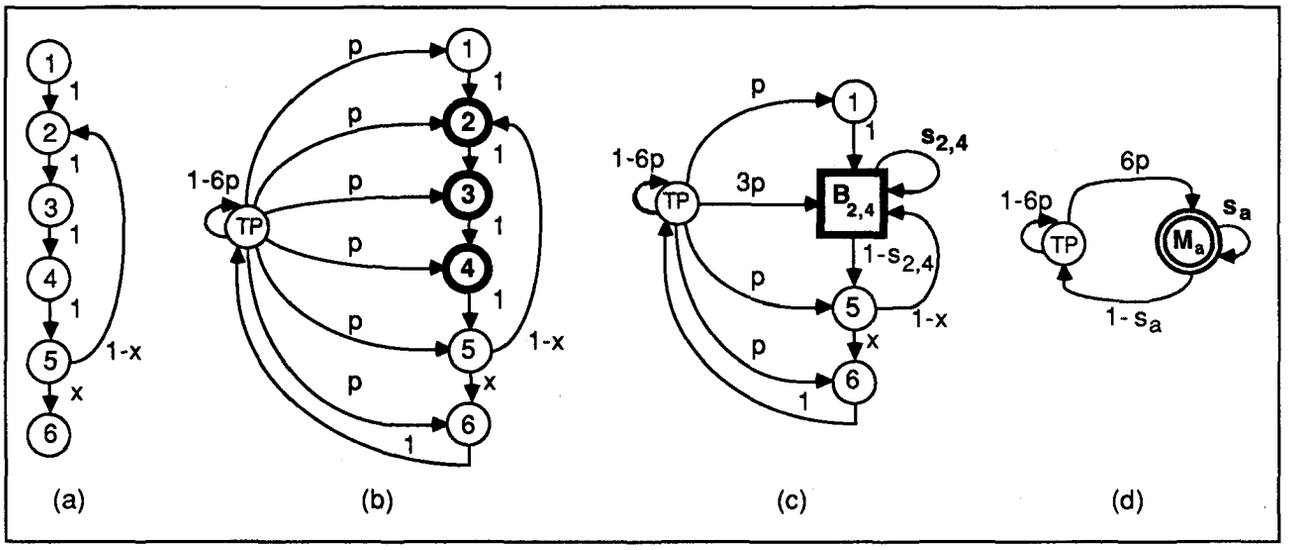


Fig. 4. — Modélisation de la carte hypothétique. (*a*) Graphe de contrôle du programme P. (*b*) D-*modèle* : $p=0{,}01$ et $x=1/32$. (*c*) S-*modèle* : $s_{2,4}=0{,}665$. (*d*) C-*modèle* (1) = C-modèle minimal : $s_a=0{,}992$.

3.2. Modèles exacts [6]

3.2.1. *Modèle détaillé :* D-*modèle*

Comme le contenu de la ROM est figé et connu, on peut représenter le programme d'application par son *graphe de contrôle* dans lequel un nœud j est associé à chaque instruction I_j. Un arc orienté allant du nœud j vers le nœud k (transition $j \to k$) indique que le µP peut exécuter l'instruction I_k après I_j dans le séquencement normal du programme. Le poids associé à cet arc correspond à la probabilité de la transition $j \to k$. Par exemple, la figure 4 *a* donne le graphe de contrôle du programme P de la figure 2 *b*. La variable x désigne la probabilité d'exécuter I_6 après I_5, *i. e.*

$$x = \text{Prob. [pas de branchement]} = \text{Prob. [Acc} = 1\,\text{F]}.$$

En substituant le graphe de contrôle du programme d'application à l'état AP de la figure 3, on obtient un modèle détaillé de la carte appelé D-*modèle*. Si le programme en ROM contient m instructions $\{I_1, I_2, \ldots, I_m\}$, le D-modèle est donc une chaîne de Markov à $(m+1)$ états : TP, 1, ..., m. Les états $\{1, 2, \ldots, m\}$ représentés par des cercles sont appelés *états simples*, chacun d'entre eux correspondant à une seule instruction. La figure 4 *b* donne un exemple de D-modèle. Une transition d'un état simple j vers TP existe lorsque l'instruction I_j peut entraîner un branchement à une adresse externe, la probabilité de ce branchement donnant celle de la transition. L'ensemble des transitions $j \to$ TP correspond à la transition AP → TP de la figure 3. La transition TP → AP de la figure 3 est remplacée par un ensemble de transitions allant de TP vers les états simples. Les probabilités de ces transitions dépendent de celles des instructions du programme de test. L'hypothèse 1 traduit le principe de mise en œuvre du test et de génération aléatoire des entrées expliqué et justifié dans [16].

Hypothèse 1 : Lorsque le programme de test effectue un branchement en ROM (transition TP → AP de la figure 3) :

(*a*) l'adresse de branchement en ROM est celle d'un mot contenant un code opération du programme d'application;

(*b*) chaque instruction du programme d'application a la même probabilité d'être adressée. □

D'après l'hypothèse 1 *a*, la transition TP → AP de la figure 3 est remplacée dans le D-modèle par m transitions TP → j ($j=1, \ldots, m$). D'après l'hypothèse 1 *b*, la probabilité p de chaque transition TP → j est, $\forall j : p=a/m$ avec $a=$ Prob. [TP → AP]. Le modèle obtenu décrit le fonctionnement correct de la carte pendant une expérience de test aléatoire, sous l'hypothèse 1.

Exemple de la carte hypothétique

Le D-modèle, représenté sur la figure 4 *b*, est une chaîne de Markov à sept états. Les calculs sont faits en supposant que, dans le programme de test :

(1) chacun des 26 codes opérations valides du µP a la même probabilité (1/26),

et

(2) chaque valeur possible des données sur 5 bits a la même probabilité $1/2^5$. □

Pendant une expérience de test, la valeur lue à l'adresse externe 0 E et chargée dans le registre accumulateur du µP par l'instruction I_1 du programme P, est fournie par le générateur aléatoire. Donc : $x=$ Prob. [Acc = 1 F] = 1/32. Six codes opérations parmi les 26 peuvent provoquer dans le programme de test un branchement en ROM. Deux d'entre eux sont des sauts conditionnels, à savoir BNE (« branchement si ≠ ») et BEQ (« branchement si = »). On obtient [17] :

$$a=0{,}06 \Rightarrow p=0{,}01 \quad \text{avec} \quad m=6.$$

3.2.2. *Modèle simplifié :* S-*modèle*

Partant du D-modèle, quatre règles de simplification conduisent à une chaîne de Markov appelée S-*modèle*, en associant un seul état appelé *état-bloc* à plusieurs instructions. Ces règles permettent de regrouper plusieurs états simples $\{i, i+1, \ldots, k-1, k\}$ dans un

état-bloc noté $B_{i,k}$ (voir *fig.* 4*c*) si et seulement si, dans le D-modèle, $\forall j=i, \ldots, k-1$: Prob. $[j \to j+1]=1$ et Prob. $[q \to j+1]=0$, $\forall q \neq j$, TP. En d'autres termes, $B_{i,k}$ représente une suite d'instructions $I_i, \ldots, I_k$ toujours exécutées séquentiellement à partir de I_i dans le programme d'application. Un S-modèle contient alors : l'état TP, des états simples q et des états-bloc $B_{i,k}$ représentés par des carrés. Trois propriétés permettent de déduire la plupart des probabilités des transitions à partir de celles du D-modèle. Mais à chaque état-bloc $B_{i,k}$ on doit associer une transition $B_{i,k} \to B_{i,k}$ de probabilité $s_{i,k}$. En effet, lorsque le µP exécute I_i le système est dans l'état $B_{i,k}$ et reste dans cet état jusqu'à l'exécution de I_k. Donc il reste dans $B_{i,k}$ avec une probabilité $s_{i,k}$ non nulle. Le théorème 1 donne la valeur de $s_{i,k}$ *appelé coefficient de réduction* associé à l'état-bloc $B_{i,k}$. Il assure la relation de *r*-équivalence (définition 1) entre le D-modèle et le S-modèle. Dans ce qui suit, $Pr[i/D]$ et $Pr[i/S]$ désignent les probabilités stationnaires d'un état i dans le D-modèle et dans le S-modèle respectivement.

Définition 1 : Le D-modèle et le S-modèle d'une carte sont dits *r-équivalents* si et seulement si :

(1) pour chaque état simple i du S-modèle on a

$$Pr[i/S]=Pr[i/D],$$

et

(2) pour chaque état-bloc $B_{i,k}$ on a

$$Pr[B_{i,k}/S]=\sum_j Pr[j/D],$$

avec

$$j=i, i+1, \ldots, k-1, k. \quad \square$$

Théorème 1 : *Le D-modèle et le S-modèle sont r-équivalents si et seulement si le* coefficient de réduction $s_{i,k}$ ($s_{i,k}<1$) est égal à :

$$s_{i,k}=\frac{n-1}{n}\cdot\frac{2.u+n.\alpha}{2.u+(n+1).\alpha}$$

où n, appelé taille du bloc, *est le nombre d'instructions regroupées dans* $B_{i,k}$; *donc* $n=k-i+1$;

$\alpha=Pr[TP].p$;

$u=\sum_q Pr[q/D]$. Prob. $[q \to i]$, $q \in U$;

U = {états $q \neq$ TP qui précèdent l'état i dans le D-modèle}. $\square$

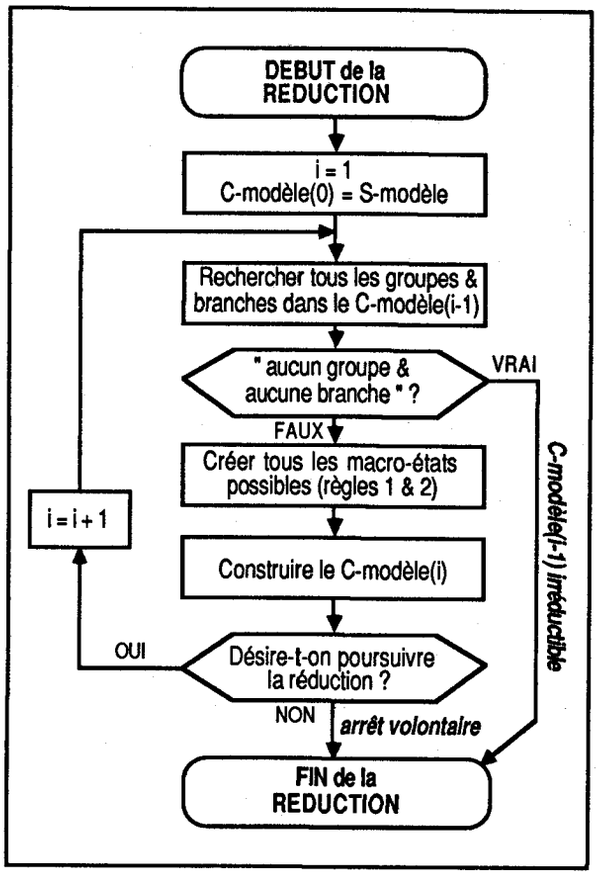
Exemple de la carte hypothétique

La figure 4*c* donne le S-modèle de la carte hypothétique. Seuls les états simples 2, 3, et 4 du D-modèle peuvent être regroupés dans un état-bloc $B_{2,4}$. Le théorème 1 donne $s_{2,4}=0{,}665$ avec : $n=3$; $u=Pr[1/D].1+Pr[5/D].(1-x)$; $Pr[1/D]=\alpha$ et $Pr[5/D]=5\alpha/x$; $x=1/32$. D'après l'hypothèse 1, on a Prob. $[TP \to B_{2,4}]=n.p=3p$.

3.3. Modèles compacts

3.3.1. *Algorithme de réduction*

La complexité du S-modèle (nombre d'états) augmente avec le nombre d'instructions de branchement dans le programme d'application et peut rester très élevée pour des cartes réelles. Pour poursuivre la réduction du modèle de la carte, on propose de créer de nouveaux états appelés *macro-états* et représentés par des doubles cercles. Chaque macro-état, noté M_j, remplace un ensemble d'états du S-modèle. Deux règles de réduction, données ci-après, permettent de telles substitutions.



5. – Modélisation compacte : algorithme de réduction.

L'*algorithme complet de réduction* est représenté sur la figure 5. C'est un processus itératif qui se déroule en plusieurs étapes, contrairement au passage du D-modèle au S-modèle qui se fait en une seule fois. Partant du S-modèle, on construit une succession de modèles appelés *modèles compacts* (C-modèles) en réduisant progressivement le nombre d'états. Le modèle obtenu à l'issue de la *i*-ième étape, noté C-modèle (i), se déduit du C-modèle ($i-1$) par création d'un ou plusieurs macro-états. Le nombre d'états représentant le programme d'application diminue ainsi à chaque étape. Dans ce processus itératif, le S-modèle correspond au C-modèle (0). Deux raisons peuvent entraîner l'arrêt des itérations :

(1) soit on estime que le C-modèle (i) contient un nombre raisonnable d'états, et on parle alors d'*arrêt volontaire*,

(2) soit le C-modèle (i) est *irréductible* s'il ne permet plus la création d'au moins un macro-état. $\square$

Un C-modèle *irréductible* est dit *minimal* s'il contient seulement deux états : TP et un seul macro-état représentant l'ensemble du programme d'application. On retrouve alors la chaîne de Markov de la figure 3. Mais la création de certains macro-états conduit à

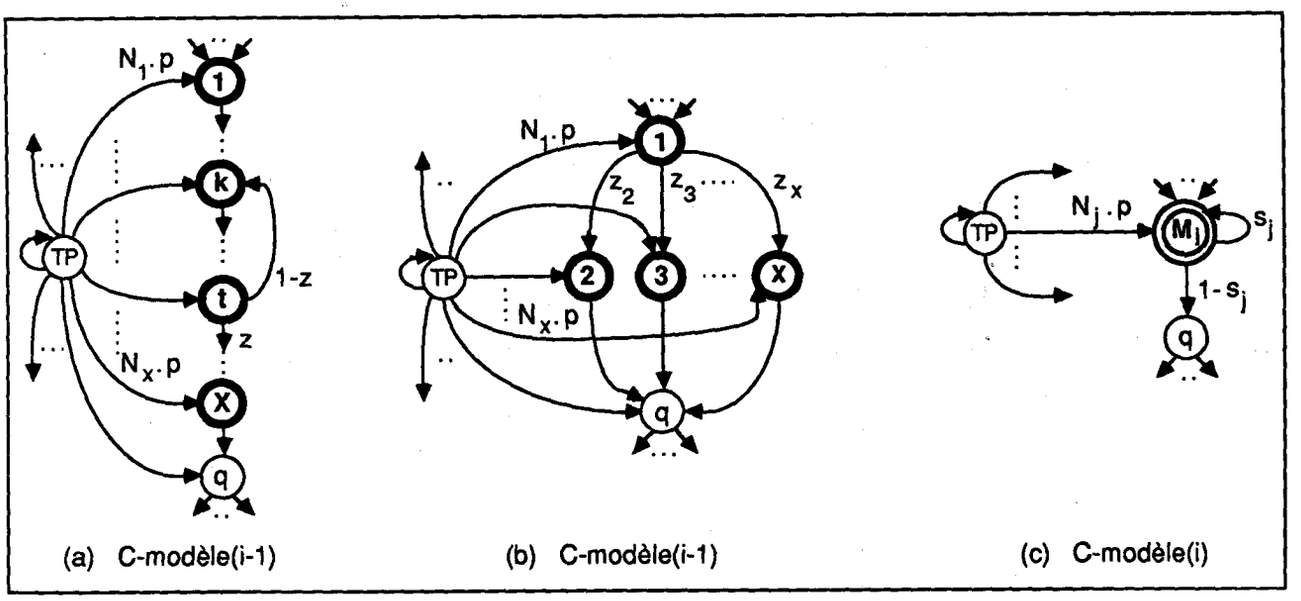


Fig. 6. — Création d'un macro-état M_j dans un C-modèle (i). Un cercle représente soit un état simple, soit un état-bloc, soit un macro-état. Pour les états $y \neq M_j$, les transitions éventuelles $y \rightarrow y$ ne sont pas indiquées. *(a) Branche* $b_{1,X}$ de X états dans un *C-modèle* $(i-1)$. *(b) Groupe* $c_{1,X}$ de X états dans un *C-modèle* $(i-1)$. *(c) Macro-état* M_j qui remplace les X états $\{1, 2, \ldots, X\}$: soit dans *(a)* $\Leftrightarrow$ *regroupement de type* 1, soit dans *(b)* $\Leftrightarrow$ *regroupement de type* 2.

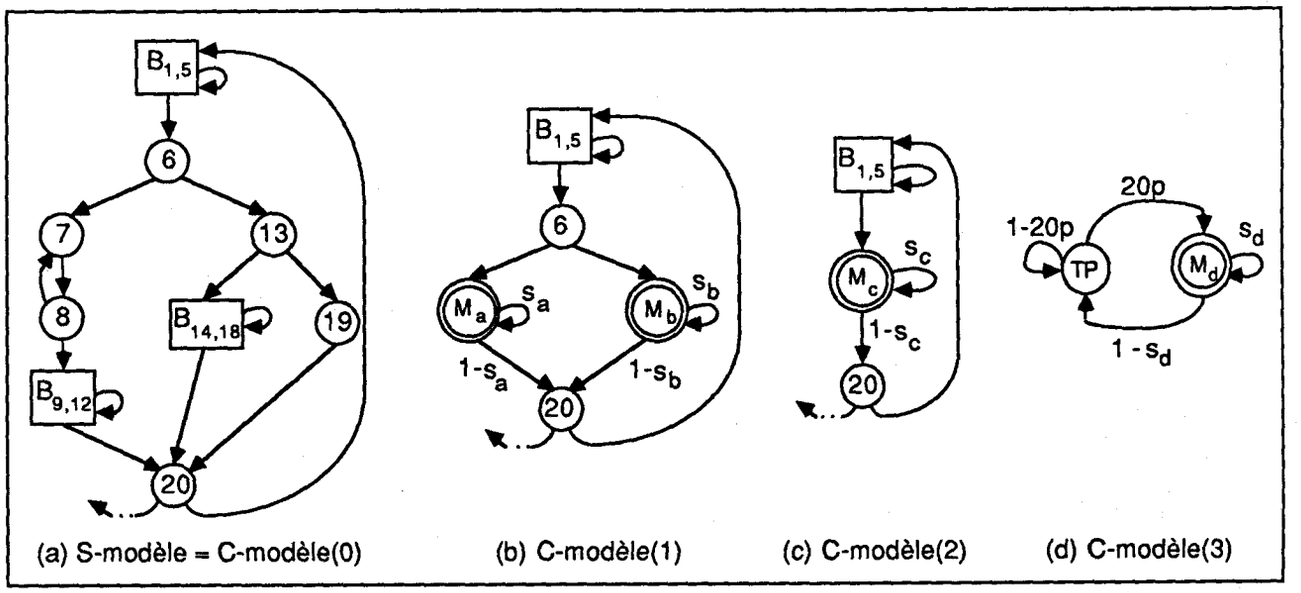


Fig. 7. — Construction des C-modèles successifs par création de macro-états M_j $(j=1, b, c, d)$. Exemple d'un programme d'application contenant 20 instructions. *(a), (b), (c)* L'état TP et les transitions issues de TP ne sont pas représentés. La transition en pointillés issue de l'état 20 conduit à TP. *(d)* Le C-modèle (3) est minimal.

des modèles approchés, comme nous le verrons au paragraphe 3.3.2. Ainsi, l'utilisateur devra faire un compromis entre la complexité du C-modèle choisi pour calculer la longueur de test et la précision du résultat final (*voir* § 4).

Dans ce qui suit, le terme général « état » désigne indifféremment les différents types d'état : état simple, état-bloc, macro-état. Les définitions 2, 3, et 4 s'appliquent à tous les modèles présentés.

Définition 2 : Un état i *précède* un état $j \neq i$ s'il existe une transition $i \rightarrow j$.

Un état i *suit* un état $j \neq i$ s'il existe une transition $j \rightarrow i$. □

Définition 3 : Une *branche* $b_{g,h}$ est une chaîne $\{g\text{-}\ldots\text{-}h\}$ d'états consécutifs pouvant contenir zéro ou une boucle. □

Cette notion est formellement définie dans [18]. La figure 6*a* montre une branche $b_{1,X} = \{1\text{-}\ldots\text{-}X\}$ incluant une boucle $\{k\text{-}\ldots\text{-}t\text{-}k\}$. Comme plusieurs états suivent l'état q, la chaîne $\{1\text{-}\ldots\text{-}X\text{-}q\}$ n'est plus une branche.

Définition 4: Un *groupe* $c_{g,h}$ est un ensemble $\{g; \ldots; h\}$ d'états non consécutifs tels que :

(1) chaque état $j \neq g$ appartenant à $c_{g,h}$ est précédé par et seulement par les états g et TP, l'état g ne précédant aucun autre état,

et

(2) tous les états $j \neq g$ appartenant à $c_{g,h}$ précèdent un seul et même état q. □

Un groupe $c_{g,h}$ est donc un ensemble d'états qui correspondent à différents chemins entre l'état g et un état q. La figure 6 *b* représente un groupe $c_{1,X}$.

Les règles de réduction ci-dessous permettent de substituer un macro-état soit à une branche, soit à un groupe, comme le montre la figure 6 *c*.

RÈGLES DE RÉDUCTION : Le C-modèle (i) est construit à partir du C-modèle ($i-1$) en substituant à l'étape i :

Règle 1 : un macro-état à chaque branche du C-modèle ($i-1$);

Règle 2 : un macro-état à chaque groupe du C-modèle ($i-1$). □

Exemple

Dans la figure 7, ces deux règles conduisent en trois itérations au modèle irréductible et minimal. En effet, la chaîne de Markov contient une branche $\{7\text{-}8\text{-}B_{9,12}\}$ et un groupe $\{13; B_{14,18}; 19\}$. A la première étape, on leur substitue respectivement les macro-états M_a et M_b dans le C-modèle(1). A la seconde étape, seul le groupe $\{6; M_a; M_b\}$ peut être remplacé par un macro-état M_c. On obtient le C-modèle(2) dans lequel le programme d'application est représenté par trois états qui forment une branche. L'étape 3 conduit alors au C-modèle(3), avec un seul macro-état M_d d'après la règle 1. Le C-modèle(3) est irréductible et minimal.

3.3.2. *Construction du C-modèle* (i)

Les probabilités des transitions dans le C-modèle (i) se déduisent de celles du C-modèle ($i-1$). Elles restent inchangées entre les états non regroupés dans un nouveau macro-état. Pour chaque macro-état M_j dans le C-modèle (i), on doit calculer (voir *fig.* 6 *c*): Prob.$[TP \to m_j]$, Prob.$[M_j \to M_j]$, Prob.$[M_j \to q]$. Pour ce faire, on associe les *trois paramètres* suivants à chaque état y ($y \neq TP$).

- N_y : nombre d'instructions écrites dans le programme d'application et représentées par l'état y;
- $Neff_y$: nombre d'instructions exécutées dans l'état y.
- S_y : Prob.$[y \to y]$ appelé coefficient de réduction associé à l'état y.

Si y est un *état simple*, alors $N_y = Neff_y = 1$ et $s_y = 0$ par construction. Si y est un *état-bloc*, alors $N_y = Neff_y =$ taille n du bloc et le théorème 1 donne la valeur de s_y. Par conséquent, ces paramètres sont connus pour tout état du S-modèle qui est utilisé comme C-modèle(0) dans l'algorithme de réduction. A chaque étape i ($i \geqq 1$), on les évalue pour les *macro-états* créés dans le C-modèle (i). La propriété 2 et les théorèmes 2, 3, 4 donnés ci-après et démontrés dans [18], permettent de les calculer en fonction des paramètres connus des états du C-modèle ($i-1$).

Par construction, les *probabilités des transitions* dans la figure 6 *c* sont données par les paramètres N_j et s_j associés à M_j, tels que :

Prob.$[TP \to M_j] = N_j . p$, d'après l'hypothèse 1;

Prob.$[M_j \to M_j] = s_j$ et Prob.$[M_j \to q] = 1 - s_j$.

N_j *associé au macro-état* M_j

Propriété 2 : Par construction, le nombre N_j d'instructions représentées par un macro-état M_j substitué à une branche ou à un groupe de X états (*fig.* 6) est :

$$N_j = \sum_{y=1,\ \ldots,\ X} N_y. \quad \square$$

Exemple de la carte hypothétique

Dans le S-modèle (*fig.* 4 *c*), les états 1, $B_{2,4}$, 5 et 6 forment une branche que l'on remplace par M_a dans le C-modèle(1) de la figure 4 *d*. Le macro-état M_a représente alors $N_a = 1+3+1+1 = 6$ instructions.

$Neff_j$ *associé au macro-état* M_j

Comme dans le cas des états-bloc, le coefficient de réduction $s_j > 0$ associé à M_j (*fig.* 6) vient du fait que la carte reste dans M_j pendant l'exécution de plusieurs instructions. Donc s_j dépend de $Neff_j$. Mais, d'après les régles de réduction données au paragraphe 3.3.1, pour un macro-état la valeur de $Neff_j$ est généralement différente de N_j. A titre d'exemple (*fig.* 6 *a*), si la branche $b_{1,X}$ contient une boucle $\{k\text{-}\ldots\text{-}t\text{-}k\}$, les instructions de la boucle peuvent être exécutées plusieurs fois ce qui conduit à $Neff_j > N_j$. Mais si M_j représente un groupe (*fig.* 6 *b*), le µP exécute seulement un sous-ensemble des N_j instructions $\Rightarrow Neff_j < N_j$. Ainsi la façon de calculer $Neff_j$ dépend de la règle de réduction utilisée.

Le nombre d'instructions exécutées dans une branche contenant une boucle varie selon le nombre d'itérations de la boucle, qui peut dépendre lui-même de la valeur de certaines variables au moment de l'exécution. De même, le nombre d'instructions exécutées dans un groupe varie selon le chemin sélectionné en fonction de la valeur prise par certaines variables au moment de l'exécution. En pratique, le théorème 2 ci-dessous permet d'évaluer une valeur moyenne du paramètre $Neff_j$, c'est-à-dire un nombre entier d'instructions qui seront « en moyenne » exécutées lors d'un passage dans le macro-état M_j associé à une branche ou à un groupe.

Théorème 2 : *Le nombre d'instructions exécutées dans un macro-état* M_j(fig. 6 *c*) *est :*

- *si* M_j *remplace la branche* $b_{1,X}$ *de la figure* 6 *a* (*règle* 1),

$$(1) \qquad Neff_j = \sum_{y=1,\ \ldots,\ X} Neff_y + \lceil A . (1-z)/z \rceil$$

avec

$1 - z = \text{Prob.}[t \to k]$;

$A = \sum_{\ddot{y}} Neff_{\ddot{y}}$, $\ddot{y} \in \{k, \ldots, t\} = \{$*états inclus dans la boucle*$\}$;

$\lceil v \rceil$ *désigne le plus petit nombre entier égal ou supérieur à* v.

- *si* M_j *remplace le groupe* $c_{1,X}$ *de la figure* 6 *b* (*règle* 2),

$$(2) \qquad Neff_j = Neff_1 + \lceil \sum_{y=2,\ \ldots,\ X} z_y . Neff_y \rceil$$

avec

$$z_y = \text{Prob.}[1 \to y],$$

$$\forall y \neq 1 \in c_{1,X} \quad \text{i.e. } y \in \{2, 3, \ldots, X\}. \quad \square$$

Dans l'équation (1), « $(1-z)/z$ » est le nombre moyen de fois où la carte passe de l'état t à l'état k, c'est-à-dire où la boucle est exécutée une fois de plus. « A » est le nombre moyen d'instructions exécutées chaque fois que le système est dans la boucle. Comme « $A.(1-z)/z$ » n'est pas toujours une valeur entière, on prend le nombre entier immédiatement supérieur, ce qui, comme nous le verrons au paragraphe 4.2, devrait conduire à un *majorant de la longueur de test.* Si la branche ne contient pas de boucle, alors $(1-z)$ est nul et l'équation (1) devient $\text{Neff}_j = \text{Neff}_1 + \text{Neff}_2 + \ldots + \text{Neff}_X$. L'équation (2) est évidente par construction.

Exemple de la carte hypothétique

Pour le macro-état M_a de la figure 4*d*, Neff_a est donné par l'équation (1) avec :

$$A = \text{Neff}_{2,4} + \text{Neff}_5 = 3 + 1 = 4;$$
$$z = x = 1/32 \quad \Rightarrow \quad (1-z)/z = 31.$$

On obtient :

$$\text{Neff}_a = \text{Neff}_1 + \text{Neff}_{2,4} + \text{Neff}_5 + \text{Neff}_6 + \ulcorner 31.4 \urcorner 6 + \ulcorner 124 \urcorner 130.$$

s_j associé au macro-état M_j

La relation de r-équivalence entre le D-modèle et le S-modèle, définie au paragraphe 3.2.2, peut être généralisée entre deux C-modèles successifs. Soit $Pr[y/i]$ la probabilité stationnaire d'un état y dans le C-modèle (i).

Définition 5 : Un macro-état M_j (*fig.* 6) est *r-équivalent* à la branche ou au groupe de X états auquel il se substitue si et seulement si :

$$(3) \qquad Pr[M_j/i] = \sum_{y=1,\ \ldots,\ X} Pr[y/i-1].$$

Le C-modèle $(i-1)$ et le C-modèle (i) sont r-équivalents si chaque macro-état créé dans le C-modèle (i) vérifie l'égalité (3). □

Théorème 3 : *Le macro-état M_j obtenu par la règle de réduction* 2 (fig. 6*b*, *c*) *est* r-équivalent au groupe $c_{1,X}$ *si et seulement si :*

$$s_j = \frac{\beta}{N_j.\alpha + u + \beta}$$

avec

$\alpha = Pr[TP].p;$

$u = \sum_v Pr[v/i-1].\text{Prob.}[v \to 1],\ v \in U;$

$U = \{$ *états* $v \neq TP$ *qui précèdent l'état* 1 *du C-modèle* $(i-1)\}$;

$$(4a) \qquad \beta = Pr[1/i-1] + \sum_{y=2,\ \ldots,\ X} s_y.Pr[y/i-1]$$

$$(4b) \qquad \beta = \left\{\frac{N_1.\alpha + u}{1-s_1}\right\}.\left\{1 + \sum_y z_y.\Omega_y\right\} + \alpha.\sum_y N_y.\Omega_y$$

où y varie de 2 à X dans les deux sommes $\sum_y$;

$$\Omega_y = s_y/(1-s_y) \quad \text{et} \quad z_y = \text{Prob.}[1 \to y] \quad \forall y \in \{2, \ldots, X\}.$$ □

La formule (4*b*) déduite de (4*a*) permet de calculer β sans avoir à évaluer les probabilités $Pr[y/i-1]$ des états y du C-modèle $(i-1)$.

Pour une *branche* $b_{1,X}$ (*fig.* 6*a*, *c*), la valeur de s_j qui assure la r-équivalence avec M_j, appelée *valeur exacte de* s_j, est donnée par une formule complexe [18]. Le théorème 4 fournit, à l'aide d'une formule plus simple, un *majorant de* s_j *dans le cas où* $b_{1,X}$ *contient une boucle*, et l'expression exacte de s_j si $b_{1,X}$ ne contient pas de boucle. La r-équivalence entre les deux modèles est alors remplacée par la R-équivalence (définition 6) qui est une relation transitive mais non symétrique. L'affectation à s_j d'une valeur supérieure à sa valeur exacte devrait conduire à l'évaluation d'un *majorant de la longueur de test*, comme nous le verrons au paragraphe 4.2.

Définition 6 : Un macro-état M_j (*fig.* 6) est R-*équivalent* à la branche ou au groupe de X états auquel il se substitue si et seulement si :

$$(5) \qquad Pr[M_j/i] \geqq \sum_{y=1,\ \ldots,\ X} Pr[y/i-1].$$

Le C-modèle (i) est R-équivalent au C-modèle $(i-1)$ si chaque macro-état créé dans le C-modèle (i) vérifie la relation (5). □

Notons que la r-équivalence est un cas particulier de la R-équivalence, lorsque la relation (5) est une égalité. Par conséquent, le C-modèle (i) est R-équivalent mais non r-équivalent au C-modèle $(i-1)$ s'il existe au moins un macro-état créé à l'étape i qui ne vérifie pas l'égalité (3).

Théorème 4: *Le macro-état M_j obtenu par la règle de réduction* 1 (fig. 6*a*, *c*) *est* R-équivalent à la branche $b_{1,X}$ *si et seulement si :*

$$s_j = \frac{2.u.(\text{Ceff}_j - 1) + \alpha.N_j.(2.\text{Ceff}_j - N_j - 1)}{2.u.\text{Ceff}_j + \alpha.N_j.(2.\text{Ceff}_j - N_j + 1)}$$

avec

$\alpha = Pr[TP].p;$

$u = \sum_v Pr[v/i-1].\text{Prob.}[v \to 1],\ v \in U;$

$U = \{$ *états* $v \neq TP$ *qui précèdent l'état* 1 *du C-modèle* $(i-1)\}$;

$$\text{Ceff}_j = \text{Neff}_j + \sum_{y=1,\ \ldots,\ X} \max(N_y - \text{Neff}_y, 0).$$ □

Exemple de la carte hypothétique

Pour le macro-état M_a de la figure 4*d*, le théorème 4 donne $s_a = 0{,}992$ avec : $u = 0$, $N_a = 6$ et $\text{Ceff}_a = \text{Neff}_a + 0 = 130$. La *valeur exacte* (celle qui assure la relation de r-équivalence) est $s_a = 0{,}989$ [18]. L'erreur relative est alors de 0,3 %. Cette erreur varie suivant la valeur de $x = \text{Prob.}[5 \to 6]$ dans la figure 4*c*. A titre d'exemple, pour $x = 0{,}5$ le théorème 4 donne $s_a = 0{,}866$ alors que la valeur exacte est 0,839 d'où une erreur relative de 3,2 %. Dans tous les résultats numériques obtenus pour d'autres programmes d'application [18], l'erreur relative reste toujours inférieure à 5 %.

4. Longueur de test aléatoire

4.1. Modélisation d'une carte défectueuse

L'activation d'une erreur latente v_f introduit une première erreur V_f dans un registre R_i du μP. L'exécu-

tion des instructions suivantes peut soit *propager* l'erreur créant ainsi de nouvelles erreurs dans d'autres registres du μP, soit *effacer* l'erreur en mettant une valeur juste dans R_i. Pour tenir compte de ces deux possibilités, propagation et suppression d'erreur, on est amené à dédoubler les états TP et AP de la figure 3. Le comportement d'une carte défectueuse pendant une expérience de test se décrit alors par la *chaîne de Markov à cinq états* représentée sur la figure 8. WT (respectivement WA) correspond à l'exécution d'une instruction du programme de test (respectivement du programme d'application) sachant qu'il y a une valeur fausse dans un ou plusieurs registres du μP. Les transitions TP → WT et AP → WA correspondent à l'activation de v_f ($e=0$ si $f \in R$, donc en particulier pour $f \in F_D$). Les transitions WA → AP et WT → TP traduisent l'effacement de la dernière valeur fausse V_f dans le μP. A la première défaillance de la carte, la chaîne est absorbée par l'*état* D (état de *détection*) qui par construction ne peut être atteint qu'à partir de WT ou WA. Les probabilités des transitions dépendent du programme d'application, mais aussi de la faute.

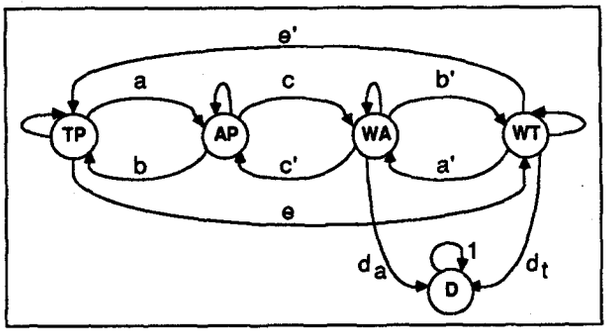


Fig. 8. – Principe de la modélisation d'une carte défectueuse.
La faute est soit en ROM, soit dans le microprocesseur.

A chaque faute f correspond donc une chaîne de Markov spécifique notée MD_f, obtenue en remplaçant chacun des états AP et WA de la figure 8 par un ou plusieurs états déduits d'un C-modèle (i) de la carte bonne. L'*algorithme de construction* d'un modèle MD_f est détaillé dans [18]. Il sera illustré par un exemple au paragraphe 4.3.

4.2. Calcul de la longueur de test

Le nombre L_f d'instructions aléatoires nécessaires pour que la faute f soit détectée avec une probabilité donnée ($1-Q_D$) s'obtient en résolvant *le système de Markov associé à la chaîne* MD_f. C'est le nombre d'instructions à exécuter pour que la probabilité de l'état D soit au moins égale à $1-Q_D$.

Dans la construction des C-modèles présentée au paragraphe 3.3.2, certaines *approximations* sont proposées pour simplifier le calcul des coefficients de réduction associés aux macro-états que l'on substitue aux branches contenant une *boucle*. Elles conduisent à surestimer, dans des proportions raisonnables, les paramètres $Neff_j$ (théorème 2) et s_j (théorème 4). Dans le C-modèle de la carte non défectueuse, les probabilités stationnaires des macro-états correspondants sont donc majorées. En conséquence, dans la chaîne absorbante MD_f, les probabilités de présence dans certains états transitoires sont plus élevées ce qui devrait conduire à un temps d'absorption plus grand, c'est-à-dire au calcul d'un *majorant de la longueur* L_f. Bien que nécessitant une démonstration plus rigoureuse, ce raisonnement semble d'autant plus vrai dans le cas des fautes $f \in F_D$ que l'on doit étudier, pour lesquelles, d'après la proposition 1, la détection ne peut pas se faire pendant l'exécution du programme d'application ($d_a = \text{Prob.}[WA \to D] = 0$ dans la figure 8). Dans tous les *exemples étudiés* [17, 18], les résultats numériques montrent que l'on obtient une valeur supérieure de L_f, ce qui en pratique garantit une qualité de détection supérieure à celle demandée ($1-Q_D$). De plus, les erreurs relatives commises sur la longueur ne dépassant pas 10%, l'augmentation du temps de test qui leur correspond reste raisonnable.

En résumé, le niveau i du C-modèle que l'on utilise pour construire MD_f détermine *la complexité de* MD_f *et la précision de* L_f. Le C-modèle(0), c'est-à-dire le S-modèle, permet d'obtenir la valeur exacte de L_f mais en résolvant un système markovien avec un grand nombre d'états. Les approximations faites dans le calcul des coefficients de réduction associés aux macro-états qui regroupent des branches avec boucle conduisent à un majorant de L_f. Le modèle MD_f déduit du C-modèle irréductible a donc un nombre minimal d'états, mais il donne une valeur plus approximative pour L_f.

Pour chaque faute $f \in F_D$ on construit MD_f, puis on calcule L_f à partir de la matrice des transitions associée à MD_f et pour des valeurs données de Q_D et du vecteur d'état initial P_0. En pratique, dans P_0 l'état TP a la probabilité 1 et tous les autres états ont une probabilité nulle (début de l'expérience de test). D'après la propriété 1 et la proposition 1 données au paragraphe 2, la longueur de test nécessaire pour détecter toute faute dans la ROM et dans le μP est alors :

$$L = \max L_f, \qquad f \in F_D.$$

4.3. Application a la carte hypothétique

Pour illustrer la construction de MD_f, reprenons l'exemple du collage à 1 du bit de poids faible dans le mot contenant l'opérande de I_1 (*voir* § 2.4). Cette faute h change l'adresse 0E en 0F (v_h). Après exécution de I_1, seul le registre Acc du μP contient une erreur V_h qui ne peut être propagée ou effacée que par des instructions du programme de test. Dans la figure 8, on a donc : $d_a = c' = 0$. La transition AP → WA correspond à l'exécution de I_1, et $e=0$. Les probabilités des transitions WT → D (observation du contenu de Acc) et WT → TP (écriture d'une valeur juste dans Acc) dépendent uniquement de celles des instructions dans le programme de test. Avec les mêmes hypothèses de calcul qu'au paragraphe 3, on obtient : $e' = 0{,}114$ et $d_t = 0{,}0385$. Pour déterminer les autres transitions (TP ↔ AP, WA ↔ WT, AP → WA), on construit la chaîne MD_h à partir soit du S-modèle (*fig.* 4*c*), soit du C-modèle(1) qui est minimal (*fig.* 4*d*).

4.3.1. *Construction de* MD_h *à partir du* S-*modèle*

Chaque état AP et WA (*fig.* 8) est remplacé par les 4 états qui représentent P dans le S-modèle. On obtient la chaîne de Markov à 11 états représentée sur la figure 9 *a* : $\{1, B_{2,4}, 5, 6\}$ correspondent à AP et $\{W1, WB_{2,4}, W5, W6\}$ à WA. La transition $1 \rightarrow B_{2,4}$ du S-modèle devient $1 \rightarrow WB_{2,4}$ dans MD_h ce qui traduit le passage de AP à WA. Le théorème 1 permet de calculer les coefficients de réduction $s_{2,4}$ et $ws_{2,4}$ à partir de la figure 9 *a*. Pour $s_{2,4}$ on a : $n=3$ et $u=Pr[5].(1-x)$ avec $Pr[5]=4\alpha/x$ et $x=1/32 \Rightarrow s_{2,4}=0{,}664$. Pour $ws_{2,4}$ c'est l'état WT qui joue le rôle de TP dans la figure 9 *a*, *i.e.* dans le théorème 1 α désigne $Pr[WT].p$ et $U=\{$états $q \neq WT$ qui précèdent $WB_{2,4}\}$. Donc $u=Pr[1].1+Pr[W1].1+Pr[W5].(1-x)$ avec $Pr\ [1]=Pr\ [TP].p$, $Pr\ [W1]=\alpha$, $Pr[W5]=(5\alpha+Pr\ [TP].p)/x$. L'équation qui exprime $Pr[TP]$ dans la chaîne MD_h conduit à la relation $Pr\ [TP].p=(e'/p).\alpha=11{,}5.\alpha$ avec $p=0{,}01$ (*voir* § 3.2.1), ce qui donne : $u=524.\alpha \Rightarrow ws_{2,4}=0{,}666$. En résolvant le système markovien associé à la figure 9 *a* on trouve :

$$L_h = 21\,376 \text{ instructions pour } Q_D = 10^{-3}.$$

4.3.2. *Construction de* MD_h *à partir du* C-*modèle minimal*

D'une façon générale, dans le C-modèle minimal un seul macro-état M_j représente le programme d'application. Pour faire apparaître la transition $AP \rightarrow WA$ (*fig.* 8) on divise M_j en *deux macro-états* M_{j1} et M_{j2} tels que l'instruction I_k modifiée par la faute f est la dernière regroupée dans M_{j1}. M_{j1} correspond aux instructions exécutées avant I_k y compris I_k, et M_{j2} aux instructions exécutées après I_k. Dans MD_f, l'état AP (respectivement WA) est donc remplacé par deux macro-états notés M_{j1} et M_{j2} (respectivement, notés WM_{j1} et WM_{j2}) ce qui conduit à une chaîne MD_f avec sept états (nombre minimum d'états pour tout modèle MD_f). Le passage de AP à WA se traduit par la transition $M_{j1} \rightarrow WM_{j2}$.

Pour la *faute h* dans la carte hypothétique, on obtient ainsi la chaîne MD_h représentée sur la figure 9 *b*. Comme *h* modifie la première instruction de P, le macro-état M_{j1} correspond seulement à l'état 1 du S-modèle, et M_{j2} (noté M) regroupe la branche $\{B_{2,4}$-5-6$\}$ du S-modèle. Les coefficients de réduction *s* et *ws* sont donnés par le théorème 4 car M et WM regroupent des branches. On obtient $s=0{,}991$ et $ws=0{,}993$. La résolution du système markovien associé à la figure 9 *b* conduit à :

$$L_h = 22\,712 \text{ instructions pour } Q_D = 10^{-3}.$$

On commet donc une erreur relative de 6,25% sur L_h.

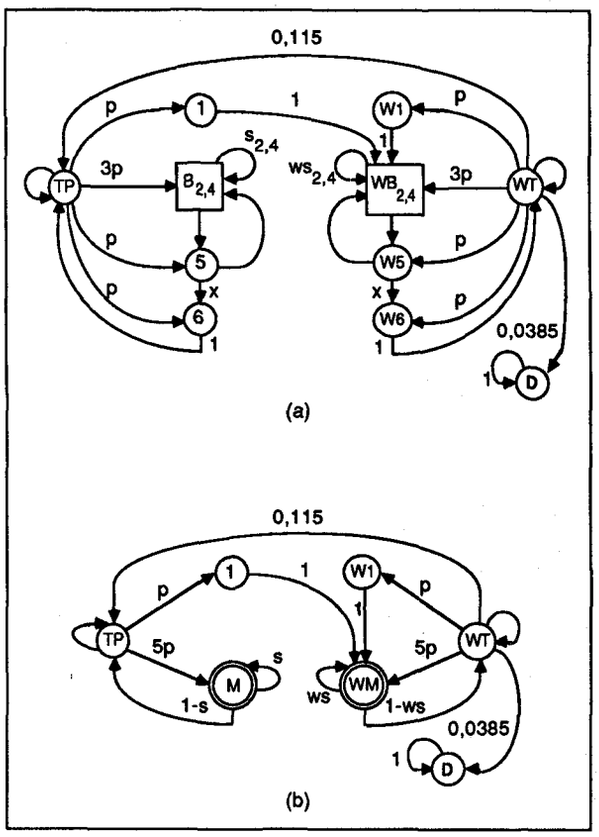


Fig. 9. — Carte hypothétique : faute h affectant l'instruction I_1 du programme P. (*a*) Chaîne de Markov MD_h *déduite du* S-*modèle* : $s_{2,4}=0{,}664$ et $ws_{2,4}=0{,}666$. (*b*) Chaîne de Markov MD_h *déduite du* C-*modèle minimal* : $s=0{,}991$ et $ws=0{,}993$.

5. Conclusion

La méthode proposée permet de calculer (un majorant de) la longueur de test aléatoire L nécessaire pour assurer une qualité de détection $1\text{-}Q_D$ donnée, en résolvant des systèmes de Markov simples. Elle a été appliquée à une *carte réelle*, avec un microprocesseur Motorola 6800 (MC 6800) et une ROM de 1 K mots de 8 bits, qui correspond à une application destinée à l'enseignement. Un programme « moniteur » est écrit dans la mémoire. Il se compose d'un programme principal de 38 instructions et de 35 sous-programmes contenant de 5 à 300 instructions. Le C-modèle irréductible est une chaîne de Markov à 20 états. Le calcul des longueurs de test nécessaires pour différentes fautes en ROM montre que les fautes les plus difficiles à détecter sont les 8 collages simples qui peuvent modifier l'opérande de la première instruction du programme principal. Ces fautes vérifient toutes la proposition 1 (§ 2.4). Le modèle MD_f, identique pour les huit collages, contient 43 états. Sa résolution conduit à $L_f = 15.10^6$ instructions pour $Q_D = 10^{-3}$. Les fautes qui modifient la première instruction dans les différents sous-programmes nécessitent des séquences de test plus courtes, de l'ordre de 5.10^6 instructions. Donc, un programme de 15.10^6 *instructions aléatoires* permet de détecter toute faute dans la ROM ou dans le MC 6800 avec une *qualité de détection de* 99,9%. Dans l'hypothèse où le testeur applique une instruction toutes les quatre microsecondes, ce qui est la vitesse de fonctionnement du testeur aléatoire pour MC 6800 présenté dans [10], la carte est testée en 1 *minute*.

Un logiciel a été mis au point pour *automatiser la modélisation des cartes et les calculs de longueurs.* Il est écrit en langage Fortran et deux versions sont

opérationnelles, l'une sur NORSK-DATA 500 et l'autre sur IBM-PC. Étant donné le graphe de contrôle du programme contenu dans la ROM et la distribution des probabilités des instructions aléatoires dans le programme de test, ce logiciel construit le S-modèle et les C-modèles successifs en évaluant les coefficients de réduction associés aux états-bloc et aux macro-états, ainsi que les matrices des transitions. D'une manière interactive, il demande ensuite des informations sur la faute f à analyser et calcule la longueur de test L_f nécessaire pour détecter f avec la qualité de détection que l'on désire.

Notons enfin que l'outil de modélisation proposé permet également d'analyser des composants intégrés du type *microcalculateurs*, qui contiennent dans un même boîtier un μP et une ROM. Une suite possible de ces travaux est l'extension de la méthode pour permettre la modélisation de cartes plus complexes, avec d'autres types de composants notamment des mémoires à lecture et écriture (RAM).

Remerciements

Ces travaux ont été effectués au *Laboratoire d'Automatique de Grenoble* (LAG). Les auteurs tiennent à remercier R. David, Directeur de l'équipe « Systèmes Logiques et Discrets » au LAG, pour ses conseils fructueux et ses encouragements tout au long de l'étude.

Manuscrit reçu le 12 *juillet* 1988, *version révisée le* 2 *février* 1989.

BIBLIOGRAPHIE

[1] P. Hansen, Functional and In-Circuit Testing Team up to Tackle VLSI in the'80s, *Electronics*, 21 avril 1981, p. 189-195.

[2] R. S. Bradley, A Three Stage Approach to LSI Board Testing, *Electronic Engineering*, 53; Part 1 : n° 651, avril 1981, p. 83-91; Part 2 : n° 654, juillet 1981, p. 43-53.

[3] J. Bateson, PCB Production Test Strategies, *Test*, 7, n° 2, mars 1985, p. 24-29.

[4] L. Renoux, GenRad revient au fonctionnel en production avec un testeur combiné, *Électronique Industrielle*, n° 130, 15 septembre 1987, p. 25-26.

[5] T. I. Sultan, Combined in-Circuit and Functional Tests: Model and Application, *Microelectronics and Reliability*, 27, n° 2, 1987, p. 279-280.

[6] Z. Abazi et P. Thevenod-Fosse, Markov Models for the Random Testing Analysis of Cards, 16*th IEEE Fault-Tolerant Computing Symposium* (*FTCS* 16), Vienne, Autriche, juillet 1986, p. 272-277.

[7] *R. David et P. Thevenod-Fosse, Test aléatoire de composants intégrés, Conférence Automatic Testing* 78, Paris, octobre 1978, 2, p. 1-15. Traduction anglaise dans *IEEE Trans. on Instrumentation and Measurement*, IM-30, n° 1, mars 1981, p. 20-25.

[8] P. Thevenod-Fosse et R. David, Random Testing of the Data Processing Section of a Microprocessor; 11*th IEEE Fault-Tolerant Computing Symposium* (*FTCS* 11), Portland, USA, juin 1981, p. 275-280.

[9] P. Thevenod-Fosse et R. David, Random Testing of the Control Section of a Microprocessor, 13*th IEEE Fault-Tolerant Computing Symposium* (*FTCS* 13), Milano, Italie, juin 1983, p. 366-373.

[10] R. David, P. Thevenod-Fosse et X. Fedi, Test aléatoire de microprocesseurs : étude théorique et expérimentations, *TSI*, 3, n° 6, 1984, p. 421-433.

[11] H. Deneux et P. Thevenod-Fosse, Random Testing of LSI Self-Checking Circuits, 2*nd International Conference on Fault-Tolerant Computing-Systems*, Bonn, RFA, septembre 1984, p. 380-390.

[12] J. Savir, G. S. Ditlow, P. H. Bardell, Random Pattern Testability, *IEEE Trans. on Computers*, C-33, n° 1, janvier 1984, p. 79-90.

[13] A. Fuentes, R. David et B. Courtois, Random Testing Versus Deterministic Testing of RAMs, 16*th IEEE Fault-Tolerant Computing Symposium* (*FTCS* 16), Vienne, Autriche, juillet 1986, p. 266-271.

[14] G. Cullmann, Initiation aux chaînes de Markov : méthodes et applications, éditions Masson, Paris, 1975.

[15] J. C. Laprie, Sûreté de fonctionnement des systèmes informatiques et tolérance aux fautes : concepts de base, *TSI*, 4, n° 5, 1985, p. 419-429.

[16] Z. Abazi et P. Thevenod-Fosse, Test aléatoire de cartes à microprocesseur : étude théorique basée sur des modèles markoviens, *TSI*, 7, n° 5, 1988, p. 477-492.

[17] Z. Abazi, Contribution à l'étude du test aléatoire de cartes à microprocesseur, *Thèse de Doctorat*, Institut National Polytechnique de Grenoble, 6 mai 1987.

[18] Z. Abazi et P. Thevenod-Fosse, La modélisation compacte, Note interne LAG 86-129, décembre 1986 révisée mai 1987.